Paquetes Genéricos

Programación de Sistemas de Telecomunicación
Informática II

Departamento de Teoría de la Señal y Comunicaciones y
Sistemas Telemáticos y Computación (GSyC)

Noviembre de 2019

Lista de enteros como TAD

- Supongamos que en un programa se necesita una lista de enteros.
- Ya hemos aprendido que:
  - es bueno programar la lista en un paquete aparte del programa principal que la usa
  - es bueno programar la lista en forma de TAD para que si hay que cambiar la implementación de la lista no sea necesario modificar el código del programa principal (cliente) que la usa. Usaremos la implementación con memoria dinámica.
- En este tema aprenderemos que también:
  - es bueno programar la lista en forma de lista genérica en vez de programar específicamente una lista de enteros.

Lista de enteros como TAD

Especificación

```
package Integer_Lists is

   type List_Type is limited private;

   procedure Add (List   : in out List_Type;
                  A_Value: in Integer);

   procedure Print_All (List: in List_Type);


private

   type Cell;
   type List_Type is access Cell;

   type Cell is record
      Value: Integer;
      Next: List_Type;
   end record;

end Integer_Lists;
```

Lista de enteros como TAD

Cuerpo

```
with Ada.Text_IO;

package body Integer_Lists is
   procedure Add (List   : in out List_Type;
                  A_Value: in Integer) is
      P_Aux : List_Type;
   begin
      P_Aux := new Cell;
      P_Aux.Value := A_Value;
      P_Aux.Next  := List;
      List := P_Aux;
   end Add;

   procedure Print_All (List: in List_Type) is
      P_Aux : List_Type;
   begin
      P_Aux := List;
      while P_Aux /= null loop
         Ada.Text_IO.Put_Line (Integer'Image(P_Aux.Value));
         P_Aux := P_Aux.Next;
      end loop;
   end Print_All;
end Integer_Lists;
```

Lista de enteros como TAD

Programa cliente

```
with Ada.Text_IO;

with Integer_Lists;

procedure Testing is

  Mi_Lista: Integer_Lists.List_Type;

begin

  Integer_Lists.Add (Mi_Lista, 39);
  Integer_Lists.Add (Mi_Lista, 22);
  Integer_Lists.Add (Mi_Lista, 19);

  Integer_Lists.Print_All (Mi_Lista);

end Testing;
```

Lista de `Floats`

- Supongamos que en el mismo programa, o en otro programa, se quiere utilizar, además de la lista de enteros, una lista de `Floats`.
- Solución (mala):
  - Se hace una copia de los ficheros `.ads` y el `.adb` de la lista de enteros con un nuevo nombre, y se retoca su código en las partes referidas al componente `Integer` de la lista, cambiándose por `Float`.

La solución es mala porque, si hay que hacer cambios en el código de la lista de enteros, hay que acordarse de repetir los cambios en el paquete de la lista de `Floats`.

Lista de Floats

Especificación

```
package Float_Lists is

   type List_Type is limited private;

   procedure Add (List   : in out List_Type;
                  A_Value: in Float);

   procedure Print_All (List: in List_Type);


private

   type Cell;
   type List_Type is access Cell;

   type Cell is record
      Value: Float;
      Next: List_Type;
   end record;

end Float_Lists;
```

Lista de Floats
Cuerpo

```
with Ada.Text_IO;

package body Float_Lists is
   procedure Add (List    : in out List_Type;
                  A_Value: in Float) is
      P_Aux : List_Type;
   begin
      P_Aux := new Cell;
      P_Aux.Value := A_Value; --no hay que cambiar nada
      P_Aux.Next  := List;
      List := P_Aux;
   end Add;

   procedure Print_All (List: in List_Type) is
      P_Aux : List_Type;
   begin
      P_Aux := List;
      while P_Aux /= null loop
         Ada.Text_IO.Put_Line (Float'Image(P_Aux.Value));
         P_Aux := P_Aux.Next;
      end loop;
   end Print_All;
end Float_Lists;
```

Lista de Floats

Programa cliente

```
with Ada.Text_IO;

with Integer_Lists;
with Float_Lists;

procedure Testing2 is

  Mi_Lista_I: Integer_Lists.List_Type;
  Mi_Lista_F: Float_Lists.List_Type;

begin

  Integer_Lists.Add (Mi_Lista_I, 39);
  Integer_Lists.Add (Mi_Lista_I, 22);
  Integer_Lists.Add (Mi_Lista_I, 19);

  Integer_Lists.Print_All (Mi_Lista_I);

  Float_Lists.Add (Mi_Lista_F, 39.23);
  Float_Lists.Add (Mi_Lista_F, 22.0);
  Float_Lists.Add (Mi_Lista_F, 19.234);

  Float_Lists.Print_All (Mi_Lista_F);
end Testing2;
```

Un paquete por cada tipo de lista que se desea implementar

Paquete Integer_Lists

```
package Integer_Lists is
   type List_Type is limited private;
   procedure Add (List: in out List_type;
                  A_Value: in Integer);
   procedure Print_All (List: in List_Type);
private
...
```

Paquete Float_Lists

```
package Float_Lists is
   type List_Type is limited private;
   procedure Add (List: in out List_type;
                  A_Value: in Float);
   procedure Print_All (List: in List_Type);
private
...
```

Programa cliente de Integer_Lists

```
with Integer_Lists:

procedure Testing is
   Mi_Lista_I: Integer_Lists.List_type;
begin
   Integer_Lists.Add (Mi_Lista_I, 39);
   Integer_Lists.Add (Mi_Lista_I, 22);
   Integer_Lists.Add (Mi_Lista_I, 19);
   Integer_Lists.Print_All (Mi_Lista_I);
end Testing;
```

Programa cliente de Float_Lists

```
with Float_Lists:

procedure Testing2 is
   Mi_Lista_F: Float_Lists.List_type;
begin
   Float_Lists.Add (Mi_Lista_F, 39.23);
   Float_Lists.Add (Mi_Lista_F, 22.0);
   Float_Lists.Add (Mi_Lista_F, 19.234);
   Float_Lists.Print_All (Mi_Lista_F);
end Testing2;
```

Reutilización de código mediante genéricos (I)

- El código de la lista (añadir elementos, mostrar todos los elementos...) es idéntico salvo por los cambios de `Integer` por `Float`.
- Programación mediante genéricos:
  - Podemos escribir la lista de forma genérica: sin precisar si los elementos de la lista son de tipo entero o de otro tipo.
  - A la hora de utilizar la lista genérica, tendremos que precisar de qué tipo concreto queremos que sean los elementos de la lista: `Integer`, `Float` o cualquier otro.
- El lenguaje Ada (Ada83) fue pionero en introducir la programación genérica.

  Hoy casi todos los lenguajes tienen este mecanismo:
  - En Ada, Java, C# se conoce como "programación mediante genéricos".
  - En C++ se conoce como "programación mediante plantillas" *(templates)*.
  - En lenguajes de tipado dinámico (Python, Ruby) este concepto no es necesario.

Reutilización de código mediante genéricos (II)

- La forma de programar una lista genérica en Ada es a través de un paquete genérico.
- Un paquete genérico indica en su especificación, antes de la palabra package, en qué es genérico el paquete:

```
generic
   type Element_Type is private;
package Generic_Lists is
   ...
end Generic_Lists;
```

Ejemplo

- Objetivo: lista genérica.

Paquete Integer_Lists

```
package Integer_Lists is
   type List_Type is limited private;
   procedure Add (List: in out List_type;
                  A_Value: in Integer);
private
...
```

Paquete Float_Lists

```
package Integer_Lists is
   type List_Type is limited private;
   procedure Add (List: in out List_type;
                  A_Value: in Float);
private
...
```

Paquete Generic_Lists

```
generic
   type Element_Type is private;

package Generic_Lists is
   type List_Type is limited private;
   procedure Add (List: in out List_type;
                  A_Value: in Element_Type);
private
...
```

Instanciar paquetes genéricos

- Para usar el paquete desde un programa principal (o desde otro paquete) es necesario instanciar el paquete, particularizándolo para un tipo genérico:

```
package Integer_Lists is new Generic_Lists (Integer);
```

- De alguna forma el tipo genérico del paquete "es como un parámetro del paquete", que se le pasa al paquete al instanciarlo con un tipo concreto.
- Un paquete genérico no es "usable" tal cual está, necesita ser instanciado. La instanciación da como resultado un paquete que sí es "usable".
  - Así el código del programa cliente usará el nombre del paquete instanciado y no el nombre del paquete genérico.

Ejemplo: Instanciación con Integer

- **El paquete genérico no es un paquete.**
- Para usar el paquete genérico desde un programa principal (o desde otro paquete) es necesario instanciar el paquete, particularizándolo para un tipo concreto.

Paquete Generic_Lists

```
generic
   type Element_Type is private;

package Generic_Lists is
   type List_Type is limited private;
   procedure Add (List: in out List_type;
                  A_Value: in Element_Type);
private
...
```

Programa cliente con lista de Integer

```
with Generic_Lists;

procedure Testing is

   package Integer_Lists is new Generic_Lists(Integer);

   Mi_Lista_I: Integer_Lists.List_type;

begin
   Integer_Lists.Add (Mi_Lista_I, 39);
   Integer_Lists.Add (Mi_Lista_I, 22);
   Integer_Lists.Add (Mi_Lista_I, 19);
   Integer_Lists.Print_All (Mi_Lista_I);
end Testing;
```

Ejemplo: Instanciación con Float

- **El paquete genérico no es un paquete.**
- Para usar el paquete genérico desde un programa principal (o desde otro paquete) es necesario instanciar el paquete, particularizándolo para un tipo concreto.

Paquete Generic_Lists

```
generic
   type Element_Type is private;

package Generic_Lists is
   type List_Type is limited private;
   procedure Add (List: in out List_type;
                  A_Value: in Element_Type);
private
...
```

Programa cliente con lista de Float

```
with Generic_Lists;

procedure Testing is

   package Float_Lists is new Generic_Lists(Float);

   Mi_Lista_F: Float_Lists.List_type;

begin
   Float_Lists.Add (Mi_Lista_I, 39);
   Float_Lists.Add (Mi_Lista_I, 22);
   Float_Lists.Add (Mi_Lista_I, 19);
   Float_Lists.Print_All (Mi_Lista_I);
end Testing;
```

Paquete genérico
Especificación

```
generic
   type Element_Type is private;
package Generic_Lists is

   type List_Type is limited private;

   procedure Add (List    : in out List_Type;
                  A_Value: in Element_Type);

   -- el procedimiento Print_All aún no podemos incluirlo

private

   type Cell;
   type List_Type is access Cell;

   type Cell is record
      Value: Element_Type;
      Next: List_Type;
   end record;

end Generic_Lists;
```

Lista genérica
Cuerpo

```
with Ada.Text_IO;

package body Generic_Lists is
   procedure Add (List    : in out List_Type;
                  A_Value: in Element_Type) is
      P_Aux : List_Type;
   begin
      P_Aux := new Cell;
      P_Aux.Value := A_Value;
      P_Aux.Next  := List;
      List := P_Aux;
   end Add;

end Generic_Lists;
```

Lista genérica
Programa cliente

```
with Ada.Text_IO;
with Generic_Lists;
with Lower_Layer_UDP;

procedure Testing is
   package LLU renames Lower_Layer_UDP;

   -- El mismo paquete genérico se instancia 3 veces para generar 3 paquetes instanciados
   package Integer_Lists is new Generic_Lists(Integer);
   package Float_Lists is new Generic_Lists(Float);
   package EP_Lists is new Generic_Lists(LLU.End_Point_Type);

   Mi_Lista_I: Integer_Lists.List_Type;
   Mi_Lista_F: Float_Lists.List_Type;
   Mi_Lista_EP: EP_Lists.List_Type;

begin

   Integer_Lists.Add (Mi_Lista_I, 39);
   Integer_Lists.Add (Mi_Lista_I, 22);
   Integer_Lists.Add (Mi_Lista_I, 19);

   Float_Lists.Add (Mi_Lista_F, 39.23);
   Float_Lists.Add (Mi_Lista_F, 22.0);
   Float_Lists.Add (Mi_Lista_F, 19.234);

   EP_Lists.Add (Mi_Lista_EP, LLU.Build("127.0.0.1", 6001));
   EP_Lists.Add (Mi_Lista_EP, LLU.Build("17.1.2.3", 9567));

   LLU.Finalize;
end Testing;
```

Lista genérica con máximo de elementos
Especificación

- Necesitamos que el tipo `List_Type` ahora sea un registro que incluya un campo para contar los elementos (en la parte privada), necesitamos una constante para el máximo de elementos (en la parte privada) y necesitamos una excepción (en la parte pública).
- Añadimos una función para que devuelva los elementos.

```
generic
   type Element_Type is private;
package Generic_Lists is

   type List_Type is limited private;

   procedure Add (List   : in out List_Type;
                  A_Value: in Element_Type);

   Full_List: exception;

   function Count (List: in List_Type) return Natural;

private

   type Cell;
   type Cell_A is access Cell;

   type Cell is record
      Value: Element_Type;
      Next: Cell_A;
   end record;

   type List_Type is record
      P_First: Cell_A;
      Total: Natural := 0;
   end record;

   Max: constant Natural := 50;

end Generic_Lists;
```

Lista genérica con máximo de elementos
Cuerpo

- En el Add se actualiza la cuenta:

```
package body Generic_Lists is

   procedure Add (List   : in out List_Type;
                  A_Value: in Element_Type) is
      P_Aux : Cell_A;
   begin
      if List.Total = Max then
         raise Full_List;
      end if;
      P_Aux := new Cell;
      P_Aux.Value := A_Value;
      P_Aux.Next  := List.P_First;
      List.P_First := P_Aux;
      List.Total := List.Total + 1;
   end Add;

   function Count (List: in List_Type) return Natural is
   begin
      return List.Total;
   end Count;

end Generic_Lists;
```

Lista genérica con máximo genérico
Especificación

- Es mejor que el máximo pueda definirse al instanciar el paquete
- Los paquetes pueden ser genéricos también en un valor:
  - Puede recibirse el valor en el paquete en modo `in`: no podrá cambiarse dentro del paquete, por lo que se comporta como una constante.
  - Puede recibirse el valor en el paquete en modo `in out`: el valor puede cambiarse dentro del paquete, por lo que debe instanciarse necesariamente con una variable.
- En nuestro ejemplo necesitamos modo `in`, y desaparece la constante de la parte private:

```
generic
   type Element_Type is private;
   Max: in Natural;
package Generic_Lists is

   type List_Type is limited private;

   ...

private
   type Cell;
   type Cell_A is access Cell;

   type Cell is record
      Value: Element_Type;
      Next: Cell_A;
   end record;

   type List_Type is record
      P_First: Cell_A;
      Total: Natural := 0;
   end record;

end Generic_Lists;
```

Lista genérica con máximo genérico
Programa cliente

- Ahora la instanciación incluye el valor máximo:

```
package Integer_Lists is new Generic_Lists(Integer, 50);
```

- La instanciación también puede hacerse en notación nombrada:

```
package Integer_Lists is new Generic_Lists(Element_Type => Integer,
                                           Max => 50);
```

Lista genérica con máximo genérico y valor por defecto

- Los valores genéricos pueden tener un valor por defecto:

```
generic
   type Element_Type is private;
   Max: in Natural := 50;
package Generic_Lists is

   ...

end Generic_Lists;
```

- Con lo que puede omitirse el valor al instanciar el paquete para usar el valor por defecto:

```
package Integer_Lists is new Generic_Lists(Integer);
```

Problema

- ¿Qué ha sido del `Print_All`?
- Recordemos su código del paquete no genérico de lista de enteros:

```
procedure Print_All (List: in List_Type) is
   P_Aux : Cell_A;
begin
   P_Aux := List.P_First;
   while P_Aux /= null loop
      Ada.Text_IO.Put_Line (Integer'Image(P_Aux.Value));
      P_Aux := P_Aux.Next;
   end loop;
end Print_All;
```

- El problema es la llamada a `Integer'Image`
- ¿En la versión genérica podríamos usar `Element_Type'Image`?
  - Poniendo en el código de `generic_lists.adb` el `Print_All` con `Element_Type'Image` aparece un error de compilación en esa línea:

```
prefix of "Image" attribute must be scalar type
```

  - No todos los tipos tienen un `'Image`. Sólo lo tienen los tipos numéricos y los enumerados.

Clases de tipos genéricos

- Según la forma de especificar el tipo genérico de un paquete:
  - se podrá usar de una determinada manera ese tipo en el cuerpo del paquete
  - se tendrá que instanciar el paquete con un tipo de determinadas características

`type T is limited private;`	T puede ser cualquier tipo NO puede usarse asignación y la comparación
`type T is private;`	T cualquier tipo con asignación y comparación, que pueden utilizarse
`type T is range <>;`	T puede ser cualquier tipo entero Puede usarse `'First`, `'Last`, `'Range` `'Image`
`type T is (<>);`	T puede ser cualquier tipo entero, carácter o enumerado. Puede usarse adicionalmente `'Pred`, `'Succ`, `'Pos`, `'Val`
`type T is digits <>;`	T puede ser cualquier tipo real

Lista genérica con Print_All

- Si se pone en la especificación de la lista genérica:

```
generic
   type Element_Type is (<>);
package Generic_Lists is
   ...
end Generic_Lists;
```

- Ahora se puede añadir el código de Print_All, pero sólo se puede instanciar el paquete para tipos enteros, caracteres y enumerados (pero no para reales o End_Points...).
- Necesitamos otra solución.

Paquetes genéricos con subprogramas genéricos

- Los paquetes también pueden ser genéricos en un procedimiento o función, que podrá invocarse desde dentro del paquete genérico cuando se necesite.
- En la especificación del paquete genérico:

```
generic
   type T is ...;
   with procedure P (...);
   with function F (...) return ...;
package Generic_Package is
   ...
end Generic_Package;
```

- Al instanciar se pasará el nombre del subprograma concreto que será llamado por el paquete, que tiene que tener los mismos parámetros que los indicados en el paquete genérico:

```
package My_Package is
            new Generic_Package(Integer, My_Proc, My_Func);
```

Lista genérica con Image genérico

Especificación

```
generic
   type Element_Type is private;
   with function Image (E: Element_Type) return String;
package Generic_Lists is

   type List_Type is limited private;

   procedure Add (List   : in out List_Type;
                  A_Value: in Element_Type);

  procedure Print_All (List: in List_Type);

private

   type Cell;
   type List_Type is access Cell;

   type Cell is record
      Value: Element_Type;
      Next: List_Type;
   end record;

end Generic_Lists;
```

Lista genérica con Image genérico

Cuerpo

```
with Ada.Text_IO;

package body Generic_Lists is

   procedure Add (List   : in out List_Type;
                  A_Value: in Element_Type) is
      P_Aux : List_Type;
   begin
      P_Aux := new Cell;
      P_Aux.Value := A_Value;
      P_Aux.Next  := List;
      List := P_Aux;
   end Add;

   procedure Print_All (List: in List_Type) is
      P_Aux : List_Type;
   begin
      P_Aux := List;
      while P_Aux /= null loop
         Ada.Text_IO.Put_Line (Image(P_Aux.Value));
         P_Aux := P_Aux.Next;
      end loop;
   end Print_All;

end Generic_Lists;
```

Lista genérica con Image genérico

Programa cliente

```
with Ada.Text_IO;
with Generic_Lists;
with Lower_Layer_UDP;

procedure Testing is
   package LLU renames Lower_Layer_UDP;

   package Integer_Lists is new Generic_Lists(Integer, Integer'Image);
   package Float_Lists is new Generic_Lists(Float, Float'Image);
   package EP_Lists is new Generic_Lists(LLU.End_Point_Type, LLU.Image);

   Mi_Lista_I: Integer_Lists.List_Type;
   Mi_Lista_F: Float_Lists.List_Type;
   Mi_Lista_EP: EP_Lists.List_Type;

begin

   Integer_Lists.Add (Mi_Lista_I, 39);
   Integer_Lists.Add (Mi_Lista_I, 22);
   Integer_Lists.Print_All(Mi_Lista_I);

   Float_Lists.Add (Mi_Lista_F, 39.23);
   Float_Lists.Add (Mi_Lista_F, 22.0);
   Float_Lists.Print_All(Mi_Lista_F);

   EP_Lists.Add (Mi_Lista_EP, LLU.Build("127.0.0.1", 6001));
   EP_Lists.Add (Mi_Lista_EP, LLU.Build("17.1.2.3", 9567));
   EP_Lists.Print_All(Mi_Lista_EP);

   LLU.Finalize;
end Testing;
```

Otros tipos genéricos para paquetes genéricos

- Arrays:

```
generic
 type Element_Type is private;
 type Index is (<>);
 type Vector is array (Index range <>) of Element_Type;
package P is
 ...
end P;
```

- Punteros:

```
generic
 type Node_Type is private;
 type P_Node_Type is access Node_Type;
package P is
 ...
end P;
```